Reinforcement Learning

Chapter 6. Temporal-Difference Learning

Presented by:

Yurou He and Ruqing Xu

March 20, 2025

What is TD Learning?

- Temporal-Difference (TD) learning is a key reinforcement learning method.
- Combines ideas from Monte Carlo (MC) and Dynamic Programming (DP).
- Learns from experience without needing a model of the environment.
- Uses bootstrapping to improve learning efficiency.

What is Bootstrapping?

- Originally means "pulling oneself up by one's bootstraps"
  – a self-starting process that proceeds without external input
- Introduced by Bradley Efron in 1979
  – Used to estimate properties of an estimator (mean, variance, confidence intervals)
  – Involves resampling with replacement from an observed dataset
- Suppose we have a small sample of heights (cm):

  160, 170, 175, 180, 165, 172, 168, 182, 176, 174
- Steps:
  1. Randomly sample (with replacement) 10 values.
  2. Compute the mean of this bootstrap sample.
  3. Repeat the process 1000 times.
  4. Use the distribution of means to estimate confidence intervals.

Bootstrapping in Reinforcement Learning

- An estimate is updated using another estimate instead of waiting for complete information.
- Used in both TD Learning and DP.
- Example: value function update in TD Learning

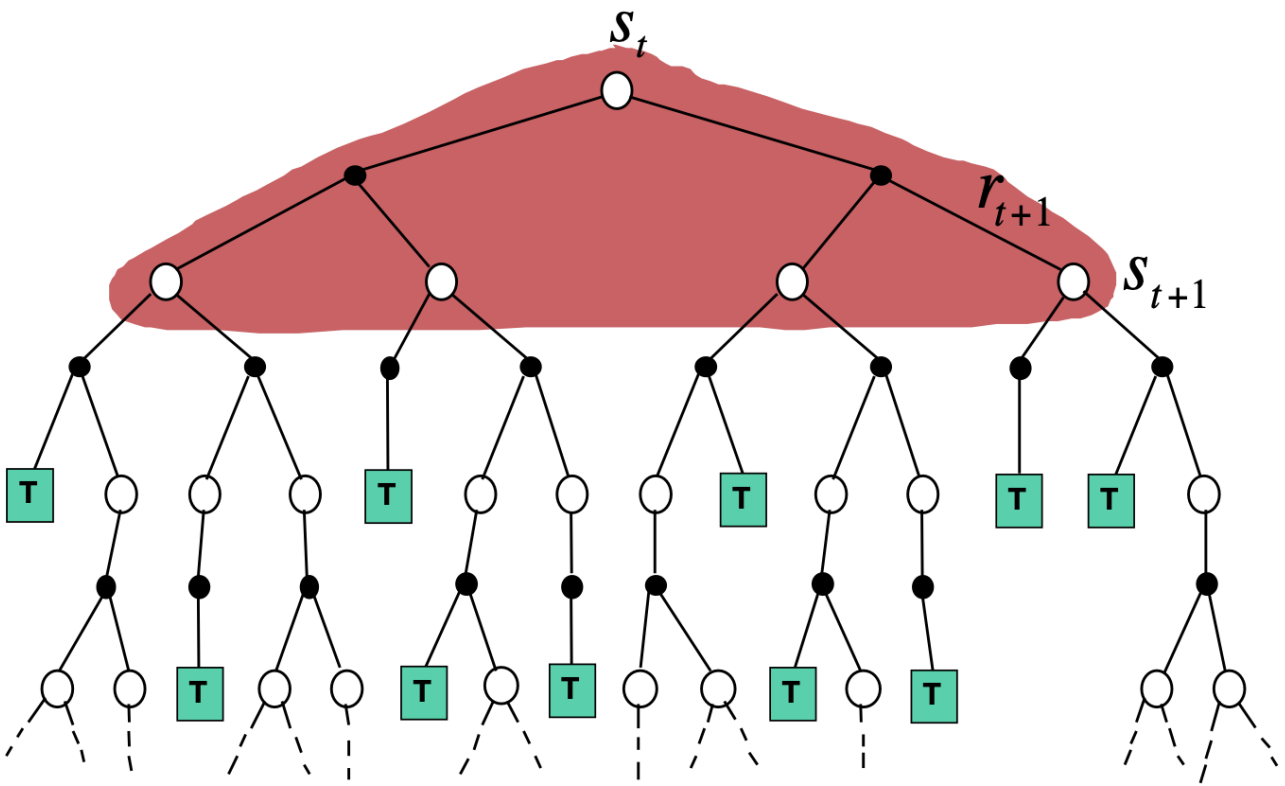
$$V(S_t) \leftarrow V(S_t) + \alpha\big(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\big)$$

  - Instead of waiting for the final return, use the estimated value of the next state.
  - Faster learning but introduces bias.

DP v.s. MC v.s. TD

cf. Dynamic Programming

$$V(s_t) \leftarrow E_\pi \left\{ r_{t+1} + \gamma V(s_t) \right\}$$

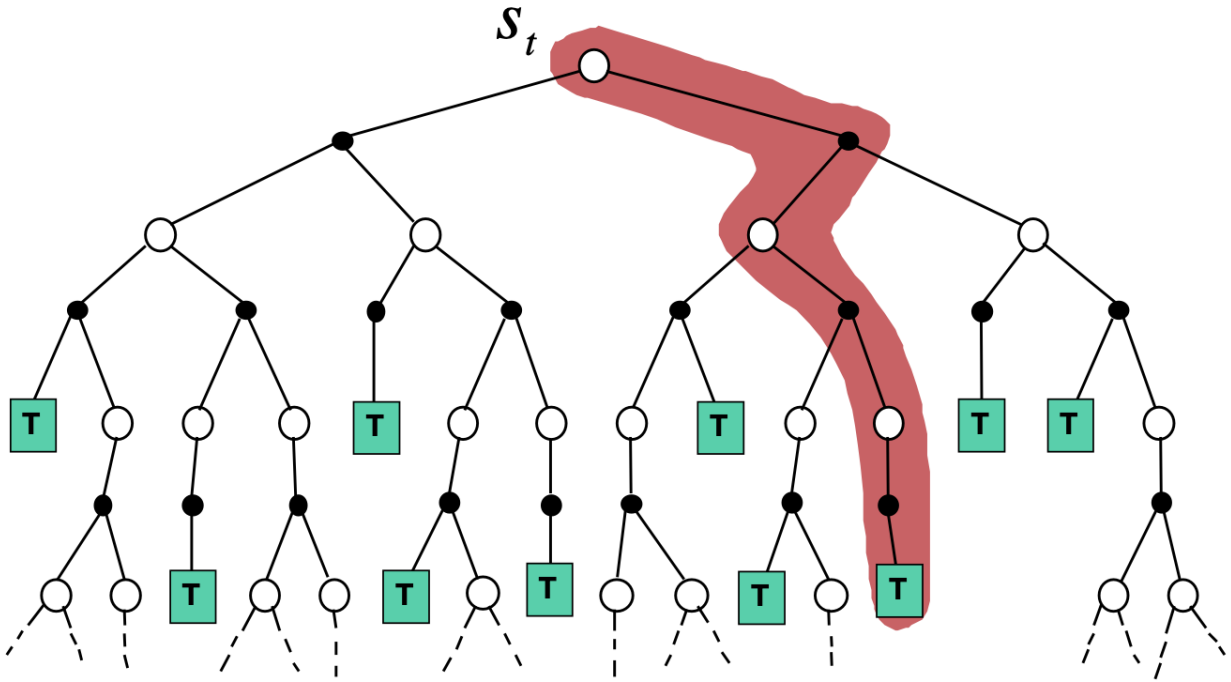


DP v.s. MC v.s. TD

Simple Monte Carlo

$$V(s_t) \leftarrow V(s_t) + \alpha\left[R_t - V(s_t)\right]$$

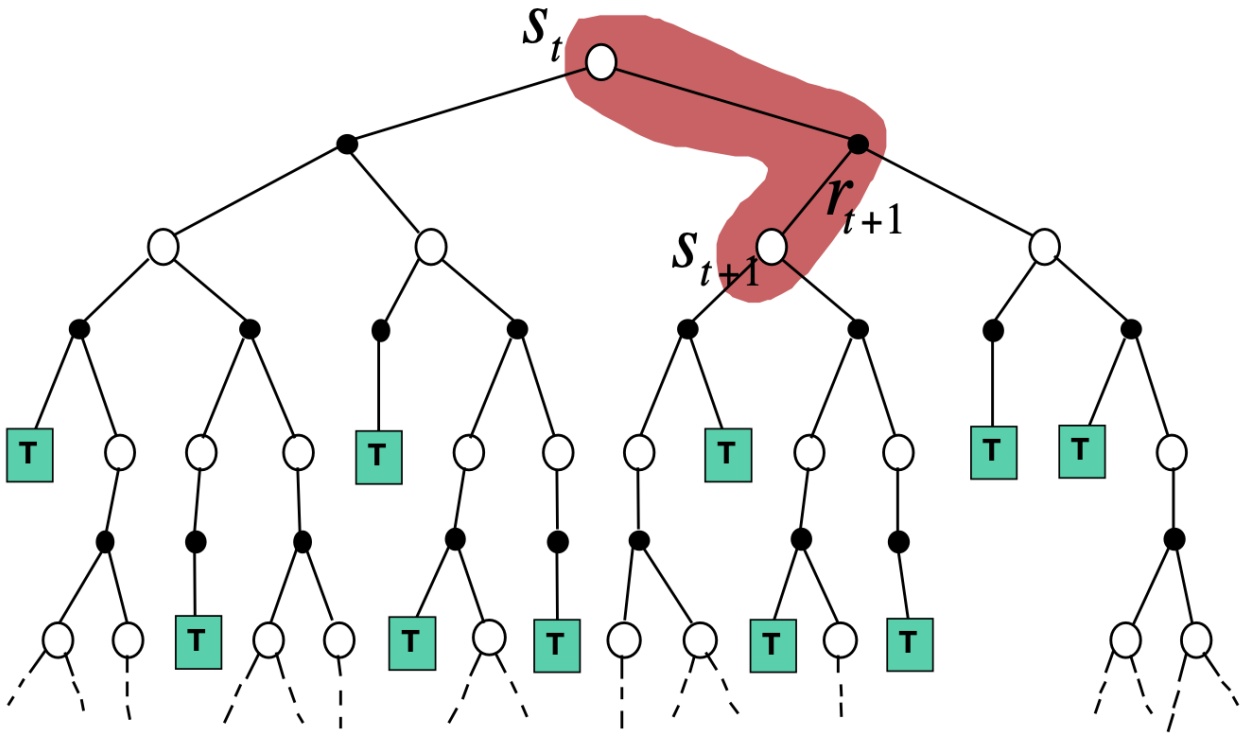
where R_t is the actual return following state s_t.



DP v.s. MC v.s. TD

Simplest TD Method

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$



When to Use TD Learning?

- When the environment model is unknown (DP).
- When episodes are long or infinite (MC).
- When computational efficiency is needed.
- When we want to use bootstrapping to propagate value updates.

Method	Model-free?	Uses bootstrapping?	Sample updates?
Dynamic Programming (DP)	No	Yes	No
Monte Carlo (MC)	Yes	No	Yes
Temporal-Difference (TD)	Yes	Yes	Yes

1. TD Prediction

Recall: Update equation in MC

$$V(S_t) \leftarrow V(S_t) + \alpha\big(G_t - V(S_t)\big)$$

Update equation with TD(0):

$$V(S_t) \leftarrow V(S_t) + \alpha\big(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\big)$$

- $V(S_t)$ - Current estimate of state value.
- R_{t+1} - Observed reward.
- γ - Discount factor.
- α - Learning rate.

Uses the current estimate $V(S_{t+1})$ instead of waiting for the final return.

1. TD Prediction

Recall: Prediction by MC

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$: # Backwards !
    $G \leftarrow \gamma G + R_{t+1}$ # Cumulative rewards counting back from T to t
    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
      Append G to $Returns(S_t)$
      $V(S_t) \leftarrow$ average$(Returns(S_t))$

Returns(St) update at most once in each episode

Prediction by TD(0):

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until S is terminal

1. TD Prediction

Example: Driving Home

How predictions are updated over time using new information:

- Initial prediction: At 6:00 PM, estimate 30 minutes to get home.
- Adjustment due to rain: At 6:05 PM, estimate is revised to 40 minutes (home by 6:40 PM).
- Highway progress: highway portion completed on time 15 minutes later, revise the total estimate to 35 minutes (home by 6:35 PM).
- Unexpected delay: get stuck behind a slow truck on a narrow road.
- Final arrival time: follow the truck until reaching a side street at 6:40 PM, then arrives home at 6:43 PM.

1. TD Prediction

Example: Driving Home

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

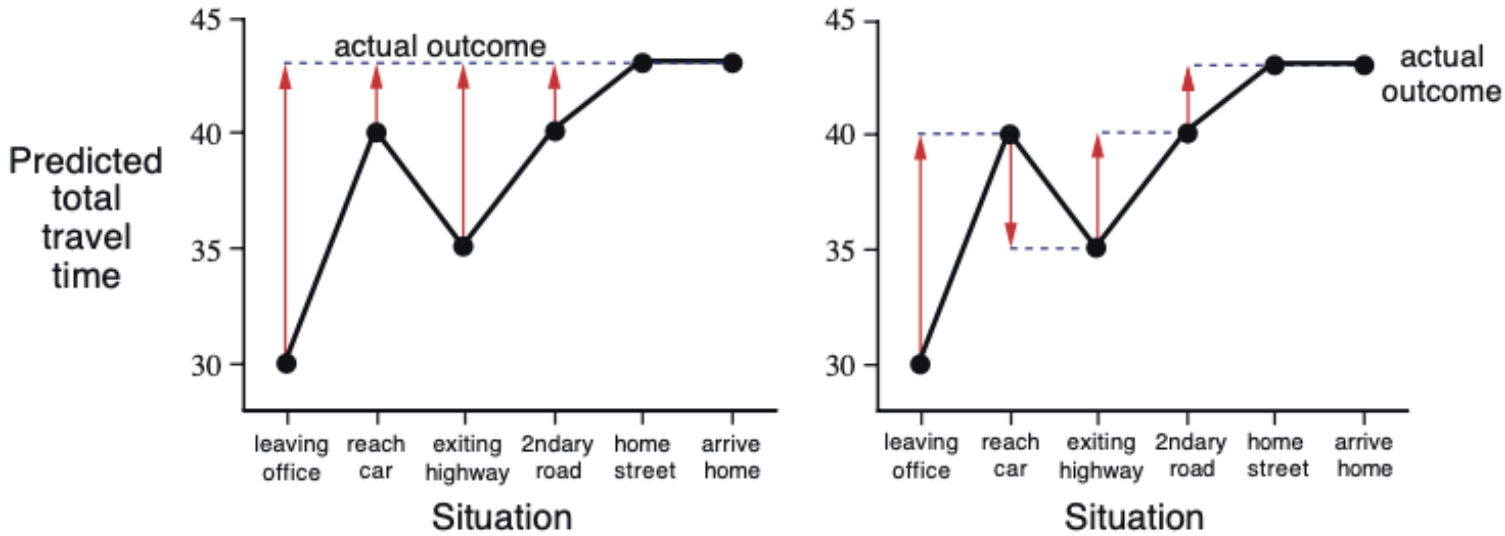


Figure 6.1: Changes recommended in the driving home example by Monte Carlo methods (left) and TD methods (right).

2. Advantages of TD

Compared to DP

- No need for an environment model $\rightarrow$ no need knowledge of reward functions or transition probabilities.

Compared to MC

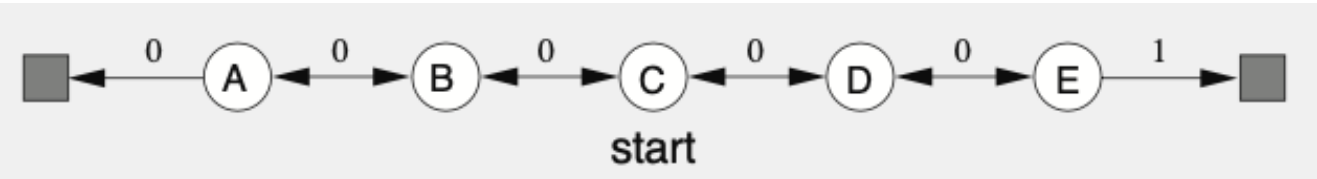
- Online and incremental updates $\rightarrow$ TD methods can update after each time step, whereas MC methods must wait until the end of an episode.
- Better suited for long episodes $\rightarrow$ Delaying updates until the episode ends can slow learning, making MC inefficient for long or infinite-horizon tasks.
- Less affected by exploration issues $\rightarrow$ MC methods may ignore or discount episodes where experimental actions occur, which can slow learning.

2. Advantages of TD

Example: Random Walk

1. Starting from C, the agent moves left or right with equal probability at each step.
2. The episode continues until it reaches the extreme left or right.
   - Reward $= +1$ if reach rightmost state
   - All other rewards $= 0$
3. Value Function: task is undiscounted, hence the value of a state equals the probability of reaching the rightmost state (reward $= +1$) from that state.

$$v_\pi(A) = \frac{1}{6}, \quad v_\pi(B) = \frac{2}{6}, \quad v_\pi(C) = \frac{3}{6}, \quad v_\pi(D) = \frac{4}{6}, \quad v_\pi(E) = \frac{5}{6}$$



2. Advantages of TD

Example: Random Walk

```python
def monte_carlo(values, alpha=0.1, batch=False):
    state = 3
    trajectory = [state]

    # if end up with left terminal state, all returns are 0
    # if end up with right terminal state, all returns are 1
    while True:
        if np.random.binomial(1, 0.5) == ACTION_LEFT:
            state -= 1
        else:
            state += 1
        trajectory.append(state)
        if state == 6:
            returns = 1.0
            break
        elif state == 0:
            returns = 0.0
            break

    if not batch:
        for state_ in trajectory[:-1]:
            # MC update
            values[state_] += alpha * (returns - values[state_])

    return trajectory, [returns] * (len(trajectory) - 1)
```

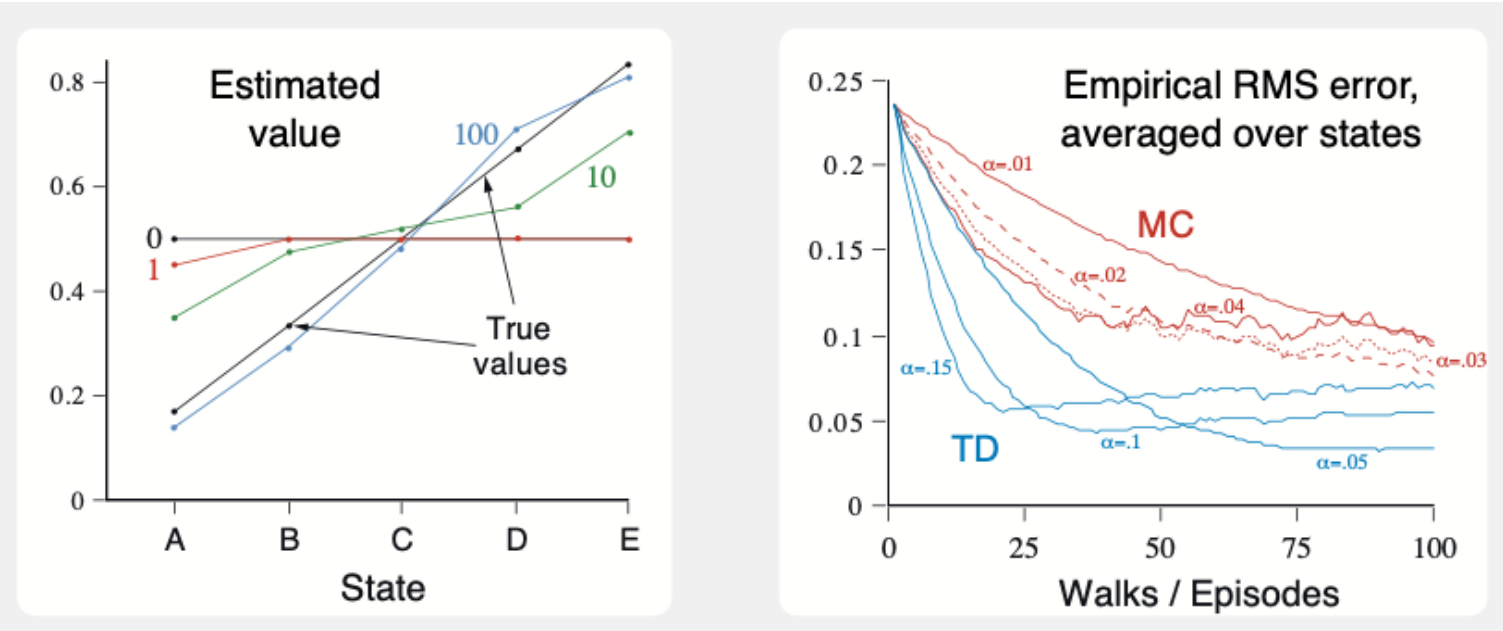
Monte Carlo

```python
def temporal_difference(values, alpha=0.1, batch=False):
    state = 3
    trajectory = [state]
    rewards = [0]

    # if end up with left terminal state, all returns are 0
    # if end up with right terminal state, all returns are 1
    while True:
        old_state = state
        if np.random.binomial(1, 0.5) == ACTION_LEFT:
            state -= 1
        else:
            state += 1

        # Assume all rewards are 0
        reward = 0
        trajectory.append(state)

        # TD update
        if not batch:
            values[old_state] += alpha * (reward + values[state] - values[old_state])
        if state == 6 or state == 0:
            break
        rewards.append(reward)
    return trajectory, rewards
```

Temporal Difference

2. Advantages of TD

Example: Random Walk



- Left: after 100 episodes, TD(0) estimates are close to the true values.
- Right: compares TD(0) and MC learning curves (RMS error) for different $\alpha = 0.1$ values.
  - The initial value is set to V(s) = 0.5 for all states.
  - TD(0) consistently outperforms MC on this task.

3. Optimality of TD(0)

Batch Updating: Train completely on a finite amount of data.
e.g., train repeatedly on 10 episodes, update the value functions until convergence.

Compute error according to TD(0),

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

but only update estimates of $V(s)$ after each complete pass through the data with the sum of all errors.

3. Optimality of TD(0)

Online TD(0)

- **Incremental Updates:** Updates value estimates immediately after each step.
- **Real-Time Learning:** Processes data as it comes.
- **Single-Pass Use:** Each experience is used once.

Batch TD(0)

- **Offline Learning:** Uses a fixed dataset.
- **Iterative Processing:** Repeatedly processes the batch until convergence.
- For any finite Markov prediction task, batch TD(0) converges to the **certainty-equivalence** estimate.

3. Optimality of TD(0)

```python
# @values: current states value, will be updated if @batch is False
# @alpha: step size
# @batch: whether to update @values
def monte_carlo(values, alpha=0.1, batch=False):
    state = 3
    trajectory = [state]

    # if end up with left terminal state, all returns are 0
    # if end up with right terminal state, all returns are 1
    while True:
        if np.random.binomial(1, 0.5) == ACTION_LEFT:
            state -= 1
        else:
            state += 1
        trajectory.append(state)
        if state == 6:
            returns = 1.0
            break
        elif state == 0:
            returns = 0.0
            break

    if not batch:
        for state_ in trajectory[:-1]:
            # MC update
            values[state_] += alpha * (returns - values[state_])

    return trajectory, [returns] * (len(trajectory) - 1)
```

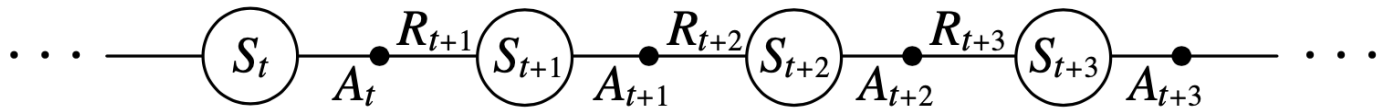
Batch in Monte Carlo

```python
# @values: current states value, will be updated if @batch is False
# @alpha: step size
# @batch: whether to update @values
def temporal_difference(values, alpha=0.1, batch=False):
    state = 3
    trajectory = [state]
    rewards = [0]

    # if end up with left terminal state, all returns are 0
    # if end up with right terminal state, all returns are 1
    while True:
        old_state = state
        if np.random.binomial(1, 0.5) == ACTION_LEFT:
            state -= 1
        else:
            state += 1

        # Assume all rewards are 0
        reward = 0
        trajectory.append(state)

        # TD update
        if not batch:
            values[old_state] += alpha * (reward + values[state] - values[old_state])
        if state == 6 or state == 0:
            break
        rewards.append(reward)
    return trajectory, rewards
```

Batch in Temporal Difference

4. Sarsa: On-policy TD Control

- For control, we need to learn the action-value function $q_\pi(s, a)$.
- We think about transitions from state-action pairs to state-action pairs.



- After every transition, do this

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma\, Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big].$$

- If S_{t+1} is terminal then $Q(S_{t+1}, A_{t+1}) = 0$.
- Named "Sarsa" because making use of $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.

4. Sarsa: On-policy TD Control

Combine this update on $Q(S, A)$ with $\epsilon-$greedy policy to the current Q.

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

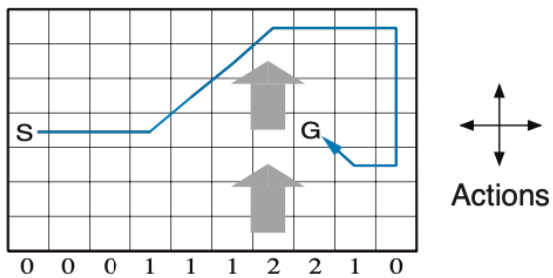
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

4. Sarsa: On-policy TD Control

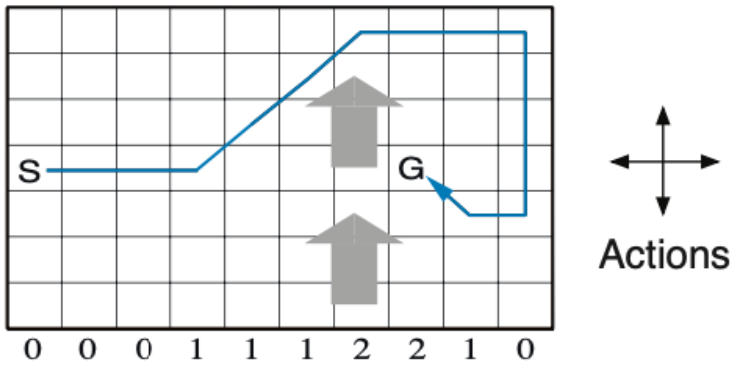
Example: Windy Gridworld

1. The agent starts at a designated start state (S) and must reach a goal state (G).
2. The agent can move in four directions: up, down, left, and right.
3. A crosswind runs through the middle of the grid, shifting the agent upward depending on the column they are in.
4. The wind strength varies across columns, as indicated by numbers at the bottom of the grid (e.g., shifting the agent up by 0, 1, or 2 cells).
5. The task is episodic and unreceived, which means that the agent receives a constant reward of 1 per step until it reaches the goal.
6. The objective is to minimize the number of steps taken to reach the goal.

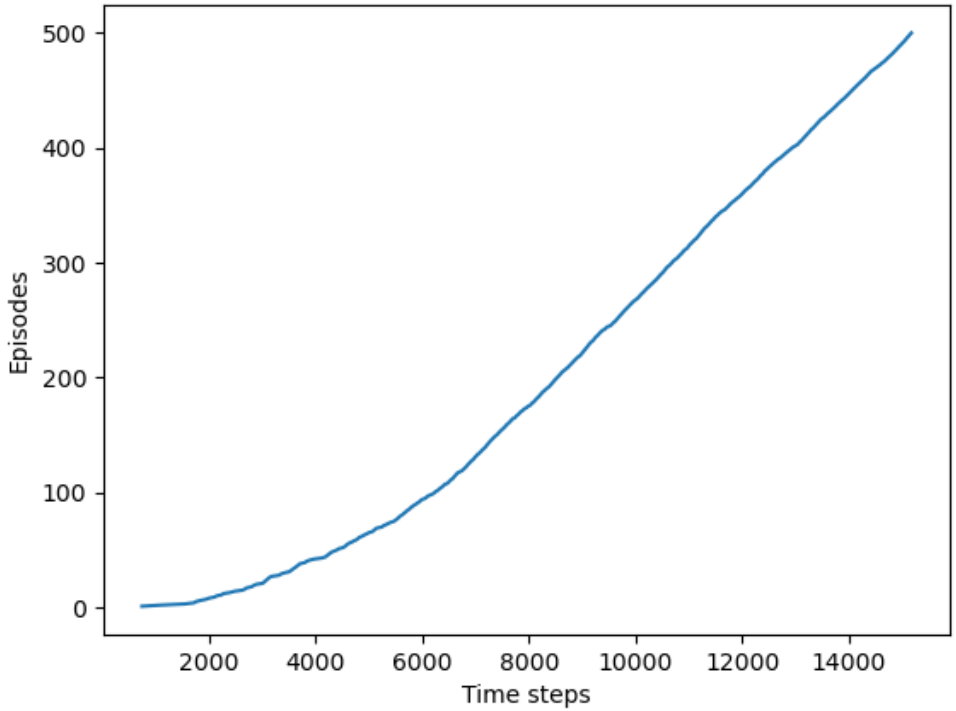


4. Sarsa: On-policy TD Control

Example: Windy Gridworld




```
Optimal policy is:
['L', 'R', 'R', 'R', 'R', 'R', 'U', 'R', 'R', 'D']
['U', 'U', 'R', 'R', 'R', 'R', 'R', 'U', 'D', 'D']
['R', 'R', 'U', 'R', 'R', 'R', 'R', 'U', 'D', 'D']
['U', 'R', 'R', 'R', 'R', 'R', 'R', 'G', 'L', 'D']
['R', 'R', 'R', 'R', 'R', 'R', 'U', 'D', 'L', 'D']
['D', 'L', 'R', 'R', 'R', 'U', 'U', 'D', 'L', 'D']
['D', 'R', 'L', 'R', 'U', 'U', 'U', 'U', 'L', 'L']
Wind strength for each column:
['0', '0', '0', '1', '1', '1', '2', '2', '1', '0']
```



- Note that Monte Carlo methods cannot easily be used here because *termination is not guaranteed* for all policies.
- The increasing slope of the graph shows that the goal was reached more quickly over time.

4. Sarsa: On-policy TD Control

Example: Windy Gridworld

```python
# Function of one episode
def episode(q_value):

    # track the total time steps in this episode
    time = 0

    # initialize state
    state = START # [3, 0]

    # choose an action based on epsilon-greedy algorithm
    if np.random.binomial(1, EPSILON) == 1:
        action = np.random.choice(ACTIONS)
    else:
        values_ = q_value[state[0], state[1], :]
        action = np.random.choice([action_ for action_, value_ in enumerate(values_) if value_ == np.max(values_)])

    # keep going until get to the goal state
    while state != GOAL:
        next_state = step(state, action)
        if np.random.binomial(1, EPSILON) == 1:
            next_action = np.random.choice(ACTIONS)
        else:
            values_ = q_value[next_state[0], next_state[1], :]
            next_action = np.random.choice([action_ for action_, value_ in enumerate(values_) if value_ == np.max(values_)])

        # Sarsa update
        q_value[state[0], state[1], action] += \
            ALPHA * (REWARD + q_value[next_state[0], next_state[1], next_action] -
                     q_value[state[0], state[1], action])
        state = next_state
        action = next_action
        time += 1
    return time
```

5. Q-learning: Off-policy TD Control

One-step Q-learning:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Q: Why is it off-policy?

5. Q-learning: Off-policy TD Control

One-step Q-learning:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

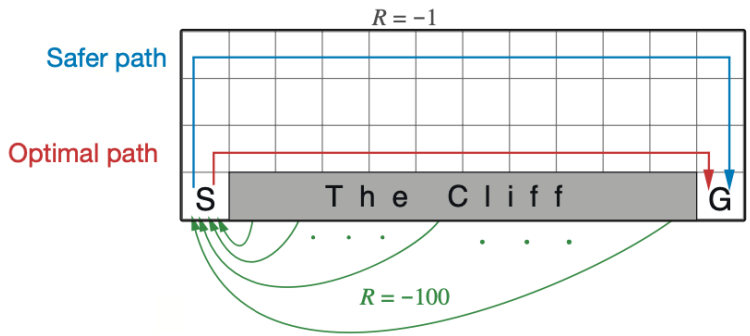
Behavioral policy

The "off-policy" part
target policy = greedy wrt Q(s,a)

5. Q-learning: Off-policy TD Control

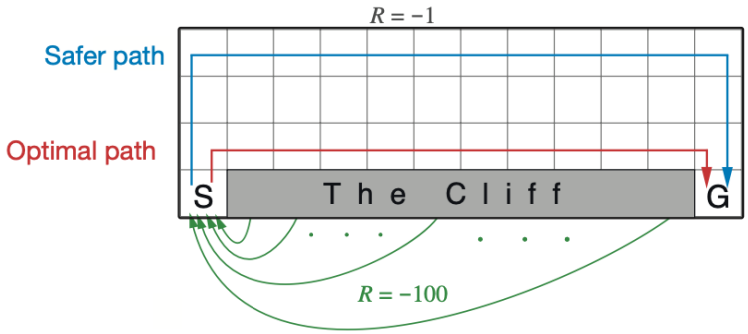
Example: Cliff Walking

1. The agent starts at the start (S) and must reach the goal (G).
2. The agent can move in four directions: up, down, left, and right.
3. The agent receives a reward of -1 for each move, except for stepping into the Cliff region.
4. The Cliff is a hazardous region—stepping into it incurs a reward of -100 and sends the agent back to the start.
5. The optimal path (in red) is the shortest but risky.
6. A safer path (in blue) avoids the cliff but takes more steps.
7. The task is episodic and undiscounted, meaning the agent must learn to reach the goal while balancing risk and efficiency.

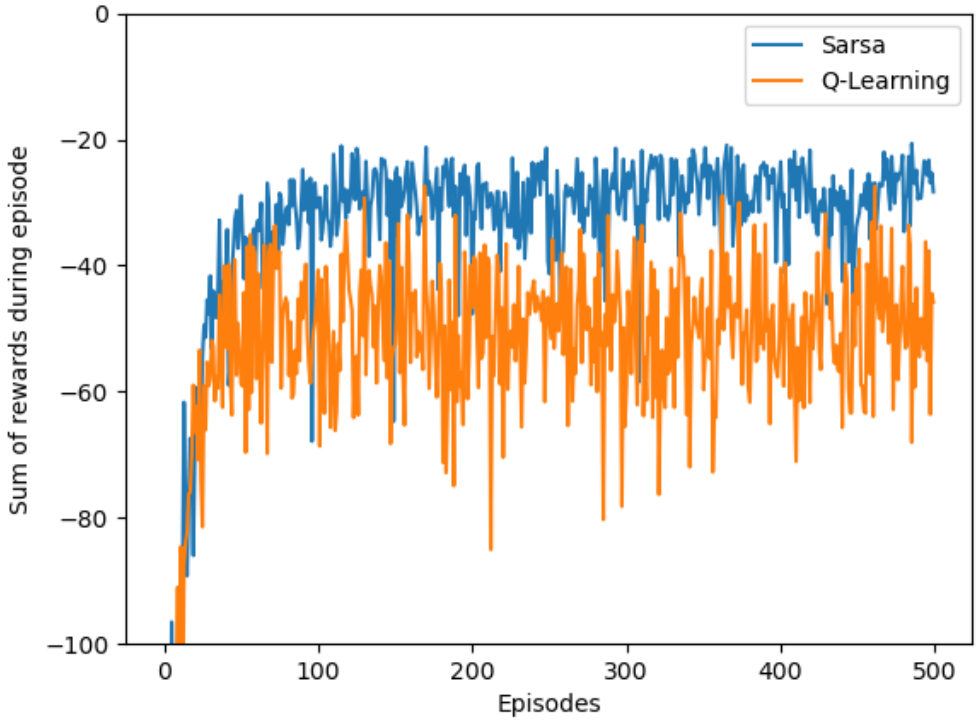


5. Q-learning: Off-policy TD Control

Example: Cliff Walking




```
Sarsa Optimal Policy:
['R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'D']
['R', 'R', 'R', 'R', 'U', 'U', 'U', 'R', 'U', 'L', 'R', 'D']
['U', 'U', 'L', 'U', 'U', 'U', 'U', 'L', 'L', 'R', 'R', 'D']
['U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'G']
Q-Learning Optimal Policy:
['R', 'U', 'R', 'L', 'R', 'U', 'D', 'R', 'D', 'R', 'R', 'D']
['R', 'D', 'D', 'R', 'R', 'D', 'D', 'R', 'R', 'D', 'D', 'D']
['R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'D']
['U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'G']
```



- Q-learning learns the optimal policy. But due to the ϵ-greedy action, the agent occasionally falls off the cliff.
- Sarsa accounts for the action selection during learning and instead learns a longer but safer path through the upper part of the grid.
- Although Q-learning ultimately learns the values of the optimal policy, its online performance is worse than that of Sarsa.

5. Q-learning: Off-policy TD Control

Example: Cliff Walking

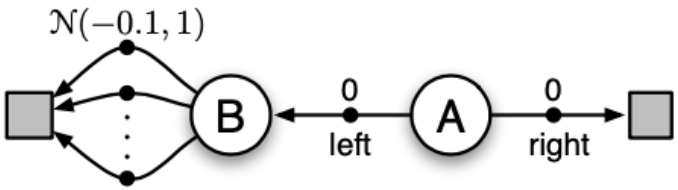
```python
# An episode with Q-Learning
# @q_value: values for state action pair, will be updated
# @step_size: step size for updating
# @return: total rewards within this episode
def qlearning(q_value, step_size=ALPHA):

    state = START
    rewards = 0.0

    while state != GOAL:
        action = choose_action(state, q_value)
        next_state, reward = step(state, action)
        rewards += reward

        # Q-Learning update
        q_value[state[0], state[1], action] += step_size * (
                reward + GAMMA * np.max(q_value[next_state[0], next_state[1], :]) -
                q_value[state[0], state[1], action])
        state = next_state
    return rewards
```

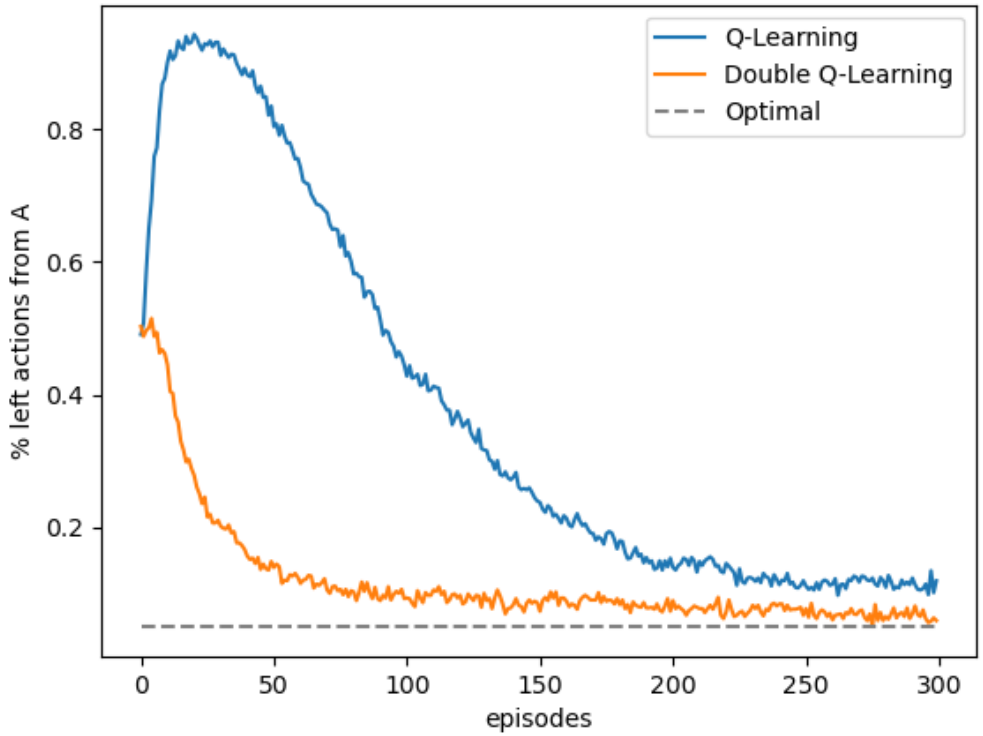
6. Maximization Bias and Double Learning

- Both Q-learning and Sarsa involve a maximization operation (explicitly so in Q-learning, $\epsilon-$greedy in Sarsa).
- However, maximization overestimates can lead to positive bias (in finite samples).
- The problem is amplified because both selection and evaluation rely on the same Q-function.
- Random Overestimates: If some actions happen to be overestimated by chance, those actions are more likely to be selected because of the $\arg\max$
- Self-Reinforcement: Once an overestimated action is chosen, the same Q-function (with the same error) is used to evaluate that action, reinforcing the overestimate and pushing the Q-value even higher.

6. Maximization Bias and Double Learning

Example





- Although worse in expected values, action *left* may be chosen more often by Q-learning.

6. Maximization Bias and Double Learning

- **Key Idea:** Maintain two independent value functions, Q^1 and Q^2, to decouple action selection from action evaluation.

1. **For each transition** (S, A, r, S')**:**
   - With probability 0.5, update Q^1:

$$A^* = \arg\max_A Q^1(S', A)$$

$$Q^1(S, A) \leftarrow Q^1(S, A) + \alpha\Big[R + \gamma\, Q^2(S', A^*) - Q^1(S, A)\Big]$$

   - Otherwise, update Q^2:

$$A^* = \arg\max_A Q^2(S', A)$$

$$Q^2(S, A) \leftarrow Q^2(S, A) + \alpha\Big[R + \gamma\, Q^1(S', A^*) - Q^2(S, A)\Big]$$

2. **Action Selection:**
   For behavior, select actions using a combination of Q^1 and Q^2 (e.g., ϵ-greedy on $Q^1 + Q^2$).

6. Maximization Bias and Double Learning

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

    else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

Summary

- TD learning is a **powerful reinforcement learning technique** that bridges MC and DP.
- Uses **bootstrapping** to update value estimates incrementally.
- **More efficient than MC** in many cases.
- **Does not require an environment model**, unlike DP.
- Key methods: **TD(0), Sarsa, Q-learning, double learning**.